

Diário de bordo 16/2

Realizado por: Cristiana Gomes

Piloto de testes: Gabriela Macieira

Antes de tudo, acedemos a este link: <https://natura.di.uminho.pt/~jj/hd2020-21/Fotos-jj/>; que nos encaminhou para um site com algumas fotos do professor. As fotos têm mais de 100 anos e ótima qualidade.



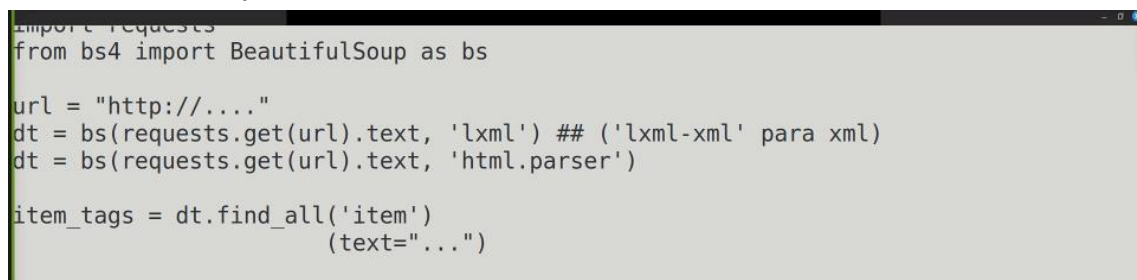
O professor recomendou outro website para visitarmos: humans in NY.

<https://www.humansofnewyork.com/>

Depois disto, demos início à aula. Retomámos o exercício da aula anterior de recorte de artigos (pasta A2). Aprofundámos BeautifulSoup.

O professor mostrou algumas cábulas de bs4:

1. Na imagem seguinte, exemplo de código onde, a partir do beautiful soup, acedemos a determinado url e extraímos as informações em html e xml. item_tags contém a função find_all que, neste caso, vai procurar os elementos que comecem na etiqueta <item> e terminem na </item>.



2. Depois, algumas funcionalidades, como:

`a.get("href")`: usado para obter o valor do atributo "href" (o link) de `<a>`.

`a.get_attribute_list("class")`: usado para obter uma lista de valores associados ao atributo de classe da tag.

`urls = [a.get("href") for a in dt.find_all("a")]`: organiza em lista todos os urls encontrados na tag `<a>`.

`.name`: dá o nome da tag.

```
for a in dt.find_all("a")
... a.get("href")
... .get_attribute_list("class")
urls=[ a.get("href") for a in dt.find_all("a")]

.contents[0].name
.children
.stripped_strings
```

3. Nesta imagem, algumas maneiras de como usar o método `find_all`.

(Encontrei um site, <https://scrapeops.io/python-web-scraping-playbook/python-beautifulsoup-findall/>, onde explica algumas das funcionalidades).

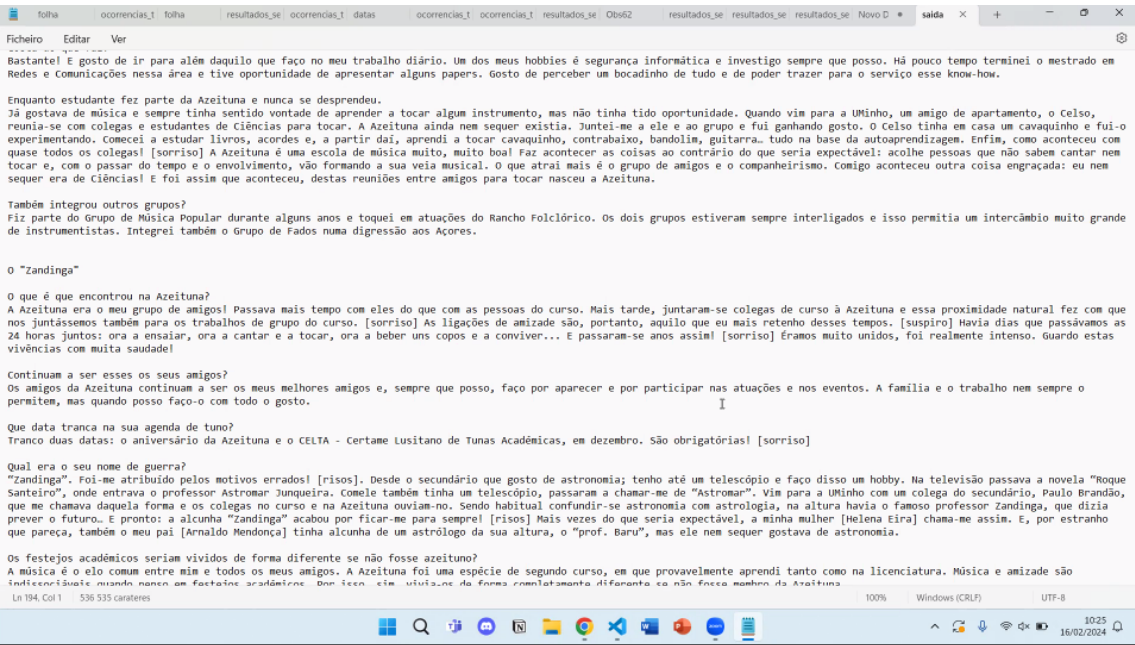
```
find_all
True      all elements
"p"
"a", "b"  <a><b>... "a" with a "b"
["a", "b"] tag a ou b
re.compile(r'\w\wbbb')
id="v"
attr={id:val}
string=re.compile()
class_ =
function dt → bool
```

4. Este print tem alguns exemplos de como alterar valores de tags:

```
modificar
del a[href]
a.unwrap()      retira a tag, mantem os conteudos
a.replaceWithChildren
a[href]=v
a.name = "f"
a.string = "v"
a.append() .extend() .insert()
a.clear()      elimina contents
a = dt.b.extract()
a.b.decompose() elimina a tag b e seu content
natura:hd2020-21$
```

Voltando ao exercício. Ponto da situação: texto sem etiquetas. Guardámos este texto limpo num ficheiro à parte, “saida.txt”

```
get.artigos.py > ...
1  from jjcli import *
2  from bs4 import BeautifulSoup as bs
3
4
5  ats= glob("nos_1146_1149/Article.aspx*")
6  print (ats)
7
8  fo= open ("saida.txt", "w", encoding="utf-8")
9
10 def proc_article(html):
11     #print (len(html)) #está a contar os caracteres de
12     a=bs(html) # cria uma árvore documental
13     art= a.find("div", id="artigo") #procura no html
14     print ("=====\n", art.get_text(), file= fo) #
15
16 for file in ats:
17     with open(file, encoding="utf-8") as f:
18         html= f.read()
19         proc_article (html)
20
```



Decidimos criar uma função extra, que colocámos no proc_article; vimos os metas, vimos os atributos, e formámos um cabeçalho. Exemplo de metas <og:title>. Criámos um loop para ir acrescentando cada um dos metas que precisávamos:

```

9
10 def proc_article(html):
11     #print (len(html)) #está a contar os caracteres de cada artigo
12     a=bs(html) # cria uma árvore documental
13     for meta in a.find_all ("meta"):
14         print (meta.get("property"), ":", meta.get("content"))
15     art= a.find("div", id="artigo") #procura no html

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS Python + - [] [X] ... ^ X

```

og:type : blog
None : None
None : IE=EmulateIE9
og:title : "Antes trabalhava-se com mais tempo"
og:url : http://www.nos.uminho.pt/Article.aspx?id=3692
og:image : http://www.nos.uminho.pt/Images/destaques/20231220165645_Conceio14.jpg
og:site_name : Jornal Online UMINHO "Antes trabalhava-se com mais tempo"
og:description : Tem 62 anos e nasceu em Braga, onde vive. A técnica superior começou a trabalhar na UMinho em 1980, ainda a Escola de Letras, Artes e Ciências Humanas era a Unidade Pedagógica de Letras e Artes. Já lá vão 43 anos!

```

Tentamos retirar o “og:”; para isso criámos um if- Se o meta fosse vazio (como na imagem de cima, “None: None”), ignorava e continuava para o meta seguinte;

Depois criámos a string “cabeçalho” onde colocámos tudo organizado, que fica da seguinte forma:

```

def proc_article(html):
    #print (len(html)) #está a contar os caracteres de cada artigo
    a=bs(html) # cria uma árvore documental
    for meta in a.find_all ("meta"):
        cabecalho = ""
        p= meta.get("property")
        if p is None:
            continue
        p = p.replace("og:", "")
        cabecalho += f"{p}: {meta.get('content')}\n"
        #print (p, ":", meta.get("content"))
    art= a.find("div", id="artigo") #procura no html
    print ("=====\n", cabecalho, art.get_text(), file= fo)

```

Problema: perderam-se imagens, que passaram a aparecer como:

```

<img src=.."
    alt=.." />

```

Tpc:

- (1) Procurar extrair e por as datas no cabeçalho
- 2) Colocar depois do Cabeçalho três (---)
- 3) Função limpeza (tirar lixo tipo voltar à página anterior)
- 4) Encontrar a lista das imagens e por no cabeçalho

Extra: Exemplo que o professor colocou no quadro; quando temos duas pastas diferentes, e queremos incluir fotos de uma específica: `glob("P*/*.jpg")`; se quero incluir fotos de P1 e P2: `glob("P**/*.jpg", _____)`

